# <https://www.codingshuttle.com/spring-boot-handbook/jwt-refresh-token-and-access-token/>

<https://medium.com/@victoronu/implementing-refresh-token-logout-in-a-spring-boot-jwt-application-b9d31de953d6>

<https://github.com/syskantechnosoft/spring-boot-refresh-token-jwt>

# Spring Boot JWT Authentication + Role-Based Authorization

This document contains a complete starter project layout and the key source files for a Spring Boot application that implements:

* JWT-based authentication (Access Token + Refresh Token)
* Role-based authorization (USER / ADMIN)
* Form-based login (Thymeleaf) and REST endpoints
* Global exception handling with @ControllerAdvice
* Thymeleaf integration for login and role-specific pages
* H2 in-memory database for quick testing

**How to use:** copy the file contents into your project structure (Maven), build and run with ./mvnw spring-boot:run or from your IDE.

## Project structure (recommended)

spring-boot-jwt-rbac/  
├─ src/main/java/com/example/security/  
│ ├─ SpringJwtRbacApplication.java  
│ ├─ config/  
│ │ └─ SecurityConfig.java  
│ ├─ controller/  
│ │ ├─ AuthController.java  
│ │ └─ HomeController.java  
│ ├─ dto/  
│ │ └─ AuthRequest.java  
│ ├─ entity/  
│ │ ├─ Role.java  
│ │ └─ User.java  
│ ├─ exception/  
│ │ └─ GlobalExceptionHandler.java  
│ ├─ repository/  
│ │ └─ UserRepository.java  
│ ├─ security/  
│ │ ├─ JwtUtil.java  
│ │ └─ JwtAuthenticationFilter.java  
│ └─ service/  
│ └─ CustomUserDetailsService.java  
├─ src/main/resources/  
│ ├─ templates/  
│ │ ├─ login.html  
│ │ ├─ home.html  
│ │ └─ admin.html  
│ └─ application.yml  
└─ pom.xml

## pom.xml

<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  
 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <groupId>com.example</groupId>  
 <artifactId>spring-boot-jwt-rbac</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>3.1.0</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
  
 <properties>  
 <java.version>17</java.version>  
 <jjwt.version>0.11.5</jjwt.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-thymeleaf</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>com.h2database</groupId>  
 <artifactId>h2</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt-api</artifactId>  
 <version>${jjwt.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt-impl</artifactId>  
 <version>${jjwt.version}</version>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt-jackson</artifactId>  
 <version>${jjwt.version}</version>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <optional>true</optional>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
</project>

## src/main/resources/application.yml

server:  
 port: 8080  
spring:  
 datasource:  
 url: jdbc:h2:mem:testdb  
 driverClassName: org.h2.Driver  
 username: sa  
 password:  
 jpa:  
 hibernate:  
 ddl-auto: update  
 show-sql: true  
jwt:  
 secret: my-very-secret-key-which-should-be-long  
 expirationMs: 600000 # 10 minutes  
 refreshExpirationMs: 1209600000 # 14 days  
  
spring.thymeleaf.cache: false  
  
logging.level.org.springframework.security: DEBUG

## Main application

package com.example.security;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class SpringJwtRbacApplication {  
 public static void main(String[] args) {  
 SpringApplication.run(SpringJwtRbacApplication.class, args);  
 }  
}

## Entity: Role + User

package com.example.security.entity;  
  
public enum Role {  
 ROLE\_USER,  
 ROLE\_ADMIN  
}

package com.example.security.entity;  
  
import jakarta.persistence.\*;  
import lombok.\*;  
import java.util.Set;  
  
@Entity  
@Table(name = "users")  
@Getter  
@Setter  
@NoArgsConstructor  
@AllArgsConstructor  
public class User {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
  
 @Column(unique = true, nullable = false)  
 private String username;  
  
 @Column(nullable = false)  
 private String password;  
  
 private String fullName;  
  
 @ElementCollection(fetch = FetchType.EAGER)  
 @Enumerated(EnumType.STRING)  
 private Set<Role> roles;  
}

## Repository

package com.example.security.repository;  
  
import com.example.security.entity.User;  
import org.springframework.data.jpa.repository.JpaRepository;  
import java.util.Optional;  
  
public interface UserRepository extends JpaRepository<User, Long> {  
 Optional<User> findByUsername(String username);  
}

## DTO: AuthRequest

package com.example.security.dto;  
  
import lombok.Data;  
  
@Data  
public class AuthRequest {  
 private String username;  
 private String password;  
}

## Service: CustomUserDetailsService

package com.example.security.service;  
  
import com.example.security.entity.User;  
import com.example.security.repository.UserRepository;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.authority.SimpleGrantedAuthority;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.stereotype.Service;  
  
import java.util.Set;  
import java.util.stream.Collectors;  
  
@Service  
public class CustomUserDetailsService implements UserDetailsService {  
  
 private final UserRepository userRepository;  
  
 public CustomUserDetailsService(UserRepository userRepository) {  
 this.userRepository = userRepository;  
 }  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 User user = userRepository.findByUsername(username)  
 .orElseThrow(() -> new UsernameNotFoundException("User not found"));  
  
 Set<GrantedAuthority> authorities = user.getRoles().stream()  
 .map(r -> new SimpleGrantedAuthority(r.name()))  
 .collect(Collectors.toSet());  
  
 return new org.springframework.security.core.userdetails.User(  
 user.getUsername(), user.getPassword(), authorities  
 );  
 }  
}

## Security utilities: JwtUtil

package com.example.security.security;  
  
import io.jsonwebtoken.\*;  
import io.jsonwebtoken.security.Keys;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
  
import java.security.Key;  
import java.util.Date;  
import java.util.Map;  
  
@Component  
public class JwtUtil {  
  
 private final Key key;  
 private final long jwtExpirationMs;  
 private final long refreshExpirationMs;  
  
 public JwtUtil(  
 @Value("${jwt.secret}") String secret,  
 @Value("${jwt.expirationMs}") long jwtExpirationMs,  
 @Value("${jwt.refreshExpirationMs}") long refreshExpirationMs  
 ) {  
 this.key = Keys.hmacShaKeyFor(secret.getBytes());  
 this.jwtExpirationMs = jwtExpirationMs;  
 this.refreshExpirationMs = refreshExpirationMs;  
 }  
  
 public String generateAccessToken(String username, Map<String, Object> claims) {  
 return Jwts.builder()  
 .setClaims(claims)  
 .setSubject(username)  
 .setIssuedAt(new Date())  
 .setExpiration(new Date(System.currentTimeMillis() + jwtExpirationMs))  
 .signWith(key)  
 .compact();  
 }  
  
 public String generateRefreshToken(String username) {  
 return Jwts.builder()  
 .setSubject(username)  
 .setIssuedAt(new Date())  
 .setExpiration(new Date(System.currentTimeMillis() + refreshExpirationMs))  
 .signWith(key)  
 .compact();  
 }  
  
 public boolean validateToken(String token) {  
 try {  
 Jwts.parserBuilder().setSigningKey(key).build().parseClaimsJws(token);  
 return true;  
 } catch (JwtException | IllegalArgumentException e) {  
 return false;  
 }  
 }  
  
 public String getUsernameFromToken(String token) {  
 return Jwts.parserBuilder().setSigningKey(key).build()  
 .parseClaimsJws(token).getBody().getSubject();  
 }  
  
 public Claims getAllClaims(String token) {  
 return Jwts.parserBuilder().setSigningKey(key).build().parseClaimsJws(token).getBody();  
 }  
}

## Security filter: JwtAuthenticationFilter

package com.example.security.security;  
  
import jakarta.servlet.FilterChain;  
import jakarta.servlet.ServletException;  
import jakarta.servlet.http.HttpServletRequest;  
import jakarta.servlet.http.HttpServletResponse;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.util.StringUtils;  
import org.springframework.web.filter.OncePerRequestFilter;  
  
import java.io.IOException;  
import java.util.List;  
import java.util.stream.Collectors;  
  
public class JwtAuthenticationFilter extends OncePerRequestFilter {  
  
 private final JwtUtil jwtUtil;  
 private final UserDetailsService userDetailsService;  
  
 public JwtAuthenticationFilter(JwtUtil jwtUtil, UserDetailsService userDetailsService) {  
 this.jwtUtil = jwtUtil;  
 this.userDetailsService = userDetailsService;  
 }  
  
 @Override  
 protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {  
 String header = request.getHeader("Authorization");  
 if (StringUtils.hasText(header) && header.startsWith("Bearer ")) {  
 String token = header.substring(7);  
 if (jwtUtil.validateToken(token)) {  
 String username = jwtUtil.getUsernameFromToken(token);  
 UserDetails userDetails = userDetailsService.loadUserByUsername(username);  
 var auth = new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());  
 SecurityContextHolder.getContext().setAuthentication(auth);  
 }  
 }  
 filterChain.doFilter(request, response);  
 }  
}

## Security configuration

package com.example.security.config;  
  
import com.example.security.security.JwtAuthenticationFilter;  
import com.example.security.security.JwtUtil;  
import com.example.security.service.CustomUserDetailsService;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;  
import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security.config.http.SessionCreationPolicy;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.security.web.SecurityFilterChain;  
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;  
  
@Configuration  
@EnableWebSecurity  
@EnableMethodSecurity  
public class SecurityConfig {  
  
 private final CustomUserDetailsService userDetailsService;  
 private final JwtUtil jwtUtil;  
  
 public SecurityConfig(CustomUserDetailsService userDetailsService, JwtUtil jwtUtil) {  
 this.userDetailsService = userDetailsService;  
 this.jwtUtil = jwtUtil;  
 }  
  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 JwtAuthenticationFilter jwtFilter = new JwtAuthenticationFilter(jwtUtil, userDetailsService);  
  
 http  
 .csrf().disable()  
 .authorizeHttpRequests((authz) -> authz  
 .requestMatchers("/login", "/refresh", "/css/\*\*", "/js/\*\*", "/").permitAll()  
 .requestMatchers("/admin/\*\*").hasRole("ADMIN")  
 .anyRequest().authenticated()  
 )  
 .formLogin((form) -> form  
 .loginPage("/login")  
 .defaultSuccessUrl("/", true)  
 .permitAll()  
 )  
 .logout((logout) -> logout.permitAll())  
 .sessionManagement((s) -> s.sessionCreationPolicy(SessionCreationPolicy.STATELESS))  
 .addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);  
  
 return http.build();  
 }  
  
 @Bean  
 public AuthenticationManager authenticationManager(AuthenticationConfiguration authConfig) throws Exception {  
 return authConfig.getAuthenticationManager();  
 }  
  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
}

## Controller: AuthController

package com.example.security.controller;  
  
import com.example.security.dto.AuthRequest;  
import com.example.security.security.JwtUtil;  
import com.example.security.entity.User;  
import com.example.security.repository.UserRepository;  
import jakarta.annotation.PostConstruct;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.Authentication;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.HashMap;  
import java.util.Map;  
  
@RestController  
public class AuthController {  
  
 private final AuthenticationManager authenticationManager;  
 private final JwtUtil jwtUtil;  
 private final UserRepository userRepository;  
 private final PasswordEncoder passwordEncoder;  
  
 public AuthController(AuthenticationManager authenticationManager, JwtUtil jwtUtil, UserRepository userRepository, PasswordEncoder passwordEncoder) {  
 this.authenticationManager = authenticationManager;  
 this.jwtUtil = jwtUtil;  
 this.userRepository = userRepository;  
 this.passwordEncoder = passwordEncoder;  
 }  
  
 // Seed two users  
 @PostConstruct  
 public void init() {  
 if (userRepository.count() == 0) {  
 User admin = new User();  
 admin.setUsername("admin");  
 admin.setPassword(passwordEncoder.encode("adminpass"));  
 admin.setFullName("Admin User");  
 admin.setRoles(Set.of(com.example.security.entity.Role.ROLE\_ADMIN));  
 userRepository.save(admin);  
  
 User user = new User();  
 user.setUsername("john");  
 user.setPassword(passwordEncoder.encode("johnpass"));  
 user.setFullName("John Doe");  
 user.setRoles(Set.of(com.example.security.entity.Role.ROLE\_USER));  
 userRepository.save(user);  
 }  
 }  
  
 @PostMapping("/login")  
 public ResponseEntity<?> login(@RequestBody AuthRequest request) {  
 Authentication auth = authenticationManager.authenticate(  
 new UsernamePasswordAuthenticationToken(request.getUsername(), request.getPassword())  
 );  
  
 var user = userRepository.findByUsername(request.getUsername()).orElseThrow();  
  
 Map<String, Object> claims = new HashMap<>();  
 claims.put("roles", user.getRoles());  
  
 String accessToken = jwtUtil.generateAccessToken(user.getUsername(), claims);  
 String refreshToken = jwtUtil.generateRefreshToken(user.getUsername());  
  
 Map<String, String> tokens = new HashMap<>();  
 tokens.put("accessToken", accessToken);  
 tokens.put("refreshToken", refreshToken);  
 return ResponseEntity.ok(tokens);  
 }  
  
 @PostMapping("/refresh")  
 public ResponseEntity<?> refreshToken(@RequestBody Map<String, String> payload) {  
 String refreshToken = payload.get("refreshToken");  
 if (refreshToken == null || !jwtUtil.validateToken(refreshToken)) {  
 return ResponseEntity.badRequest().body("Invalid refresh token");  
 }  
 String username = jwtUtil.getUsernameFromToken(refreshToken);  
 var user = userRepository.findByUsername(username).orElseThrow();  
 Map<String, Object> claims = new HashMap<>();  
 claims.put("roles", user.getRoles());  
 String newAccessToken = jwtUtil.generateAccessToken(username, claims);  
 return ResponseEntity.ok(Map.of("accessToken", newAccessToken));  
 }  
}

Note: This controller implements a simple REST /login that authenticates with username/password and returns access & refresh tokens. For form-based login (Thymeleaf) the SecurityConfig formLogin() handles showing the login page and authentication.

## Controller: HomeController (Thymeleaf pages)

package com.example.security.controller;  
  
import org.springframework.stereotype.Controller;  
import org.springframework.ui.Model;  
import org.springframework.web.bind.annotation.GetMapping;  
import java.security.Principal;  
  
@Controller  
public class HomeController {  
  
 @GetMapping("/")  
 public String home(Principal principal, Model model) {  
 model.addAttribute("username", principal == null ? "anonymous" : principal.getName());  
 return "home";  
 }  
  
 @GetMapping("/admin")  
 public String admin(Principal principal, Model model) {  
 model.addAttribute("username", principal == null ? "anonymous" : principal.getName());  
 return "admin";  
 }  
  
 @GetMapping("/login")  
 public String login() {  
 return "login";  
 }  
}

## Global Exception Handler

package com.example.security.exception;  
  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.ControllerAdvice;  
import org.springframework.web.bind.annotation.ExceptionHandler;  
import org.springframework.web.bind.annotation.ResponseBody;  
  
@ControllerAdvice  
public class GlobalExceptionHandler {  
  
 @ExceptionHandler(Exception.class)  
 @ResponseBody  
 public ResponseEntity<?> handleAll(Exception ex) {  
 return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR).body(ex.getMessage());  
 }  
}

## Thymeleaf templates

### login.html

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <title>Login</title>  
</head>  
<body>  
<h2>Login</h2>  
<form th:action="@{/login}" method="post">  
 <div>  
 <label>Username</label>  
 <input type="text" name="username" />  
 </div>  
 <div>  
 <label>Password</label>  
 <input type="password" name="password" />  
 </div>  
 <div>  
 <button type="submit">Login</button>  
 </div>  
</form>  
</body>  
</html>

### home.html

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <title>Home</title>  
</head>  
<body>  
<h2>Welcome, <span th:text="${username}">user</span></h2>  
<p><a th:href="@{/admin}">Admin Page</a></p>  
</body>  
</html>

### admin.html

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<head>  
 <meta charset="UTF-8">  
 <title>Admin</title>  
</head>  
<body>  
<h2>Admin Page - Hello <span th:text="${username}">admin</span></h2>  
</body>  
</html>

## Notes & Improvements

1. **Token storage:** In production, store refresh tokens securely (DB or revocation list) and implement refresh token rotation.
2. **Secret management:** Do not store JWT secret in plain text. Use Vault or environment variables.
3. **Claims:** Keep minimal claims in the access token. Roles can be included but avoid sensitive data.
4. **CSRF:** For form login, CSRF is useful. For REST APIs using JWT, CSRF can be disabled with care.
5. **Exception details:** Improve GlobalExceptionHandler to return structured error JSON with codes.
6. **Logout / Token revocation:** Add endpoint to revoke refresh tokens.
7. **Validation & DTOs:** Add request validation for login and refresh.

## Quick Run

1. Create the maven project and paste files as shown.
2. Run the application and open http://localhost:8080/login for the Thymeleaf login.
3. Use seeded users: admin/adminpass (admin), john/johnpass (user).
4. Use POST /login (JSON) to obtain access and refresh tokens (REST):

curl -X POST http://localhost:8080/login -H "Content-Type: application/json" -d '{"username":"john","password":"johnpass"}'

1. Call protected APIs with header Authorization: Bearer <accessToken>.